RENDERING TRANSLUCENT LAYERS IN A DISPLAY SYSTEM

Inventors:
Ralph T. Brunner
Peter Graffagnino

CROSS REFERENCE TO RELATED APPLICATIONS

[0001] This application is a continuation of U.S. Patent Application Serial Number 09/309,171, filed on May 10, 1999, the disclosure of which is hereby incorporated by reference.

BACKGROUND OF THE INVENTION

1. Field of the Invention

[0002] The present invention relates generally to computer-implemented display systems, and more particularly to a system and method of rendering translucent and complex-shaped overlapping layers in a display system.

2. Description of Background Art

[0003] Many existing display systems are capable of compositing two or more display elements to generate a final image. In such systems, display

elements often include overlapping layers, such as for example in a windowing system for a graphical user interface wherein on-screen elements, such as windows, may be moved around and placed on top of one another.

[0004] Rendering and displaying an image having two or more overlapping layers presents certain problems, particularly in determining how to render that portion of the image where the layers overlap. When the overlapping layers are opaque, the graphics system need only determine which layer is on "top", and display the relevant portion of that layer in the final image; portions of underlying layers that are obscured may be ignored. However, when overlapping layers are translucent, more complex processing may be called for, as some interaction among picture elements (pixels) in each overlapping layer may take place. Accordingly, some calculation may be required to overlay the image elements in order to derive a final image.

[0005] Compositing techniques for performing these calculations are known in the art. See, for example, T. Porter et al., in "Compositing Digital Images", in *Proceedings of SIGGRAPH '84*, 1984, pp. 253-59. Generally, however, such techniques are directed toward compositing only two layers at a time. When more than two layers are to be composited, a number of separate operations must be performed in order to generate the final image. This is generally accomplished by compositing image elements in a bottom-up ap-

proach, successively combining each new layer with the results of the compositing operations performed for the layers below.

[0006] This step-by-step compositing approach has several disadvantages. If the image is constructed in the frame buffer, on-screen flicker may result as the system writes to the frame buffer several times in succession. Alternatively, the image may be constructed in an off-screen buffer, thus avoiding on-screen flicker; however, such a technique requires additional memory to be allocated for the buffer, and also requires additional memory reads and writes as the final image is transferred to the frame buffer.

[0007] In addition, step-by-step generation of the final image may result in poor performance due to the large number of arithmetic operations that must be performed. Writing date to a frame buffer is particularly slow on many computers; therefore, conventional systems which write several layers to the frame buffer in succession face a particularly severe performance penalty.

[0008] Finally, such a technique often results in unnecessary generation of some portions of image elements that may later be obscured by other image elements.

[0009] Conventionally, arbitrarily shaped windows and layers are accomplished by dividing a window into rectangular areas and/or blocking out some portions of a rectangle to make a rectangular layer appear to be arbitrarily

shaped. Such techniques often result in additional processing time for areas of the windows that may have no effect on the final image.

[0010] What is needed is a system and method for rendering translucent layers, which avoids the above-referenced deficiencies and allows compositing of multiple layers without causing screen flicker and without requiring additional off-screen buffers. What is further needed is a rendering system and method that improves the performance of prior art systems when generating images having multiple overlapping translucent layers. What is further needed is a rendering system and method that avoids unnecessary generation of image element portions that will be obscured by other image elements in the final image.

SUMMARY OF THE INVENTION

[0011] In accordance with the present invention, there is provided a rendering system and method that facilitates efficient compositing of translucent and complex-shaped overlapping layers. An alpha channel is provided for each layer, in order to allow transparency characteristics to be specified on a pixel-by-pixel basis within the layer. Rendering is performed in front-to-back sequence, so that the invention is able to avoid unnecessary computations on layer portions that do not contribute to the final image. In addition, in one embodiment, compositing is made more efficient by first

subdividing the image area into rectangles, wherein the set of overlapping layers (and their ordering) for all points within a given rectangle is constant. Compositing operations can then be performed on each rectangle separately, so as to avoid complex and time-consuming bounds-checking operations inside the innermost loop of the processing operation.

[0012] The present invention renders two or more overlapping layers, each having an alpha channel, into a destination bitmap. The destination bitmap may be the frame buffer of a video system, for example. For each pixel in the destination bitmap, the present invention processes the layers having a corresponding pixel in front-to-back sequence, accumulating color and alpha values. Once the accumulated alpha channel reaches full opacity, the present invention stops working on that pixel; any pixels lying beneath the stopping point would be completely obscured (since full opacity has been reached) and need not be processed.

[0013] The present invention thus avoids reading pixel data that corresponds to pixels that do not contribute to the final image. This economy in image processing results in improved performance of the graphics system. In addition, if the destination bitmap is the frame buffer, flickering effects are reduced because each pixel in the destination bitmap is written at most once. The present invention also enables easy implementation of visual effects such as

fading in or fading out of windows, as well as shadows, glow effects, and the like.

[0014] Additional performance enhancement is achieved, in one embodiment, by tagging the topmost layer that has changed since the last update. If the front-to-back compositing operation stops before reaching the tagged layer, it is not necessary to write the result to the output bitmap (the frame buffer), since the output bitmap already contains the same data, from the last update. The present invention can thus avoid unnecessary write operations in such a situation, and thereby improve system performance, particularly when output is being written to the frame buffer.

[0015] In one embodiment, additional optimization is achieved by initially subdividing the destination bitmap into a number of tiles, each tile containing the same stack of layers to be rendered. By performing such subdivision, the present invention is able to make determinations as to which layers may contribute to the final image on a tile-by-tile basis rather than on a pixel-by-pixel basis. This simplifies the bounds-checking process, and minimizes the number of calculations that need be made in the innermost loop of the process, thus improving overall performance.

[0016] The present invention can be implemented, for example, in a windowing system to provide efficient rendering of overlapping translucent windows in an improved graphical user interface.

BRIEF DESCRIPTION OF THE DRAWINGS

[0017] Fig. 1 is a block diagram of the overall architecture of an embodiment of the present invention.

[0018] Fig. 2 is a diagram showing an example of the output of the present invention.

[0019] Fig. 3 is a diagram showing top-down processing of a pixel according to the present invention.

[0020] Fig. 4 is a diagram showing an example of tile subdivision according to one embodiment of the present invention.

[0021] Fig. 5 is a diagram showing an example of tile subdivision according to one embodiment of the present invention.

[0022] Fig. 6 is a flowchart showing the operation of one embodiment of the present invention.

[0023] Fig. 7 is a flowchart showing color and alpha merging according to one embodiment of the present invention.

[0024] Fig. 8 is a flowchart showing tile subdivision according to one embodiment of the present invention.

[0025] Fig. 9 is a flowchart showing a method of joining tiles according to one embodiment of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0026] Referring now to Fig. 1, there is shown a block diagram of the overall architecture of one embodiment of the present invention. In this embodiment, system 100 is implemented on a conventional personal computer having a central processing unit and/or graphics processor 101 and a display 105 such as a cathode ray tube (CRT) or liquid crystal display (LCD) device. The steps of the present invention may be stored as software in an operating system or application software for execution by processor 101 in the computer, as is known in the art. For example, system 100 may be implemented on a Macintosh® computer, from Apple Corporation, running the MacOS operating system, and the software may form part of a Window Server module within the operating system.

[0027] Layer data 103 represent source images to be composited for display on display 105. For example, layer 103 may include bitmap representations of windows, graphical user interface (GUI) elements, photographs, drawings, animations, and the like. In one embodiment, each layer 103 may be of any arbitrary shape and size, and may contain translucent, transparent, and opaque regions, in any combination desired. In one embodiment, each layer 103 contains a plurality of data structures 106-108 for storing color (RGB) values, alpha values, and fade values, as appropriate. Any number of layers 103 may be provided for processing by system 100.

[0028] Graphics processor 101 may be implemented as a conventional central processing unit (CPU) of the computer, or it may be a dedicated processor for graphics operations. Accumulator 102, in one embodiment, is a register or memory location for temporary storage of values being calculated. As will be described below, more than one accumulator 102 may be provided, and in one embodiment the present invention is able to process several pixels in parallel by employing multiple accumulators 102.

[0029] Frame buffer 104 is an area of memory, such as random access memory (RAM), that is reserved for storage of display data, as is known in the art. In one embodiment, frame buffer 104 is connected to display 105 so that display 105 outputs the contents of frame buffer 104 after processor 101 has written results to frame buffer 104. In other embodiments, processor 101 may write images to other areas of memory (not shown), or to disk (not shown), or to some other storage device, rather than to frame buffer 104.

[0030] Referring now to Fig. 2, there is shown an example of the output of the present invention. Three layers 103 are provided as input to system 100 for compositing. Output image 204, as may be shown on display 105, contains all three layers 103 overlapping one another. As can be seen from the Figure, layers 103 can be of any arbitrary shape, and may contain opaque, transparent, and/or translucent regions, in any combination. In one embodiment, all layers are provided as rectangular regions, so that arbitrarily-shaped images can be

represented as rectangular layers having both transparent and non-transparent regions. Compositing of transparent layers or areas is a trivial operation, as the underlying image is unaffected by the presence of an overlapping transparent area.

[0031] In one embodiment, each layer 103 contains a plurality of pixels, each pixel having a color value. The color value of each pixel represents a color according to a conventional color-encoding scheme (such as a red-green-blue, or RGB color encoding scheme). In one embodiment, each pixel also has an alpha value representing a relative degree of transparency (or opacity) for that pixel. Such a technique facilitates full control over the transparency of each layer on a pixel-by-pixel basis, permitting translucency effects as well as arbitrarily-shaped images.

[0032] As shown in the example of Fig. 2, the present invention is able to generate output that combines several translucent and complex-shaped windows. Thus, the invention can be employed to enable an on-screen windowing system containing a graphically rich user interface in which overlapping elements or windows are translucent, so that the user can see underlying elements.

[0033] Referring now to Fig. 3, there is shown an example of top-down processing of a pixel according to the present invention. In one embodiment, the present invention processes each pixel of the image separately. In another

embodiment, several pixels may be processed in parallel using a number of accumulators, as will be described below.

[0034] For each pixel 301 in the image, one or more layers 103 may contribute to the color value of pixel 301. Generally, layers 103 that may contribute to the color value are those layers 103 which contain image data corresponding to the physical coordinates associated with the image pixel being processed. Before processing a pixel 301, accumulator 102 is reset. Then, for each contributing pixel 301 in one of layers 103, the present invention accumulates the color and alpha values of the contributing pixel 301 to derive a final value for the image pixel.

[0035] As is known in the art, each layer 103 has a relative "vertical" position. The vertical position of a layer 103 is a conceptual representation of the order in which the layers are stacked on one another. Thus, if the final image portrays layer B as a window that partially obscures a portion of layer A, layer B would be considered "on top" of layer A.

[0036] In one embodiment, the invention reads pixel data from layers 103 in top-down order, with the topmost layer 103 being accumulated first. As each pixel's data is read, it is merged with the current value of accumulator 102 using a compositing method as described in more detail below. In one embodiment, the invention accumulates alpha values as well as color values. If the accumulated alpha value indicates full opacity before all layers 103 have been pro-

cessed, processing for the current pixel may be terminated, since any layers 103 beneath the current layer would be completely obscured by the layers 103 that have already been processed. In this way, the present invention is able to avoid reading and processing pixel data that does not contribute to the final image.

[0037] Once the invention has read all layers 103 that may contribute to pixel 301, or once full opacity has been reached, the value in accumulator 102 is output. In one embodiment, the value is written directly to frame buffer 104. Since the accumulation of color values occurs in accumulator 102, only one write to frame buffer 104 is required, so that excessive flickering of the on-screen display (due to multiple writes to frame buffer 104) is avoided. Furthermore, since accumulation occurs in accumulator 102, no off-screen buffer for temporary storage of an image is required, as accumulator 102 output can be written directly to frame buffer 104.

[0038] Referring now to Fig. 6, there is shown a flowchart of the operation of the invention according to one embodiment. The steps shown in Fig. 6 may be performed for each pixel in the final image for which multiple layers have corresponding pixels. Processor 101 initializes 602 accumulator 102 with initial color and alpha values. In one embodiment, accumulator 102 is implemented as a register in processor 101. In another embodiment, two or more accumulators 102 are provided, so that the steps of Fig. 6 are performed concurrently for a number of pixels.

[0039] Processor 101 selects 603 a layer 103 from among those layers having a pixel corresponding to the position of the pixel being rendered. In one embodiment the topmost layer 103 is selected, so as to implement top-down processing, though in other embodiments, any layer 103 may be selected.

[0040] Processor 101 identifies a pixel in layer 103 having a position corresponding to the image pixel being rendered, and merges 604 the color value of the identified pixel with the current value of accumulator 102. In one embodiment, processor 101 also merges the alpha value of the identified pixel with an alpha value stored in accumulator 102.

[0041] Processor 101 then determines 605 whether any remaining layers can contribute to the color value of the image pixel being rendered. In one embodiment, where top-down processing is being used, this determination is made by checking whether the accumulated alpha value indicates that full opacity has been reached. If full opacity has been reached, no underlying pixels can contribute to the final color value of the image pixel. If no remaining layers can contribute, the accumulated color value in accumulator 102 is output 606. In one embodiment, the value is output to frame buffer 104 for display on display 105, though output to other memory locations or storage devices may instead be performed.

[0042] If in step 605, accumulator 102 does not indicate full opacity, or some other determination is made that additional layers 103 need be processed,

system determines 607 whether any additional layers exist that have pixels corresponding to the image pixel being rendered. If not, the accumulated color value in accumulator 102 is output 606.

[0043] If in step 607, additional layers 103 remain, processor 101 selects 608 another layer 103 from among those layers having a pixel corresponding to the position of the image pixel being rendered. In one embodiment the topmost remaining layer 103 is selected, so as to implement top-down processing, though in other embodiments, any remaining layer 103 may be selected. Processor 101 then returns to step 604.

[0044] Steps 604, 605, and 607 are repeated until either full opacity has been reached, or some other determination is made that any remaining layers 103 cannot contribute to the color value of the image pixel being processed, or no layers 103 remain.

[0045] The steps of Fig. 6 may be repeated for each pixel in the image, as needed. Pixels may be processed in any order.

[0046] In one embodiment, in step 603 processor 101 loops through each successive layer (starting with the top layer) until it finds a layer containing a non-transparent pixel for the pixel position being rendered. If the first non-transparent pixel is found to be fully opaque, the result can be immediately written to the frame buffer without performing any merging operations. If the

pixel is not fully opaque, blending and merging operations continue with step 604.

[0047] In one embodiment, processor 101 merges color and alpha values in step 604 according to known compositing techniques, such as those described in T. Porter et al., in "Compositing Digital Images", in *Proceedings of SIGGRAPH '84,* 1984, pp. 253-59. Referring now to Fig. 7, there is shown a method of merging color and alpha values. Opacity and color values are initialized 701 by setting values to zero. A first layer is selected 702. If full opacity has been reached (indicated by an opacity value reaching a predetermined threshold, such as 1.0), the process ends 708.

[0048] As long as full opacity has not been reached, and additional layers exist, steps 703 through 707 are repeated. In 704, the fade value for the pixel is determined, using the equation:

$$\gamma = \delta_n \bullet \alpha_n (x,y) \bullet (1-\alpha) \qquad \text{(Eq. 1)}$$

where:

δ_n is the overall fade value for layer n;

$\alpha_n (x,y)$ is the opacity (alpha) value of the current pixel in layer n; and

α is the accumulated opacity (alpha) value.

[0049] In 705, the color values for the pixel are adjusted by the fade value, and added to the accumulator, using the equations:

$$r = \tilde{a} \bullet r_n(x,y) + r \qquad \text{(Eq. 2)}$$

$$g = \tilde{a} \bullet g_n(x,y) + g \qquad \text{(Eq. 3)}$$

$$b = \tilde{a} \bullet b_n(x,y) + b \qquad \text{(Eq. 4)}$$

where:

$r_n(x,y)$, $g_n(x,y)$, $b_n(x,y)$ are color values of the current pixel in layer n; and

r, g, b are the accumulated color values.

In 706, the fade value for the pixel is added to the opacity accumulator, by:

$$\alpha = \tilde{a} + \alpha \qquad \text{(Eq. 5)}$$

[0050] In 707, the next layer is selected (n is incremented).

[0051] In the embodiment represented by Fig. 7, the invention stores input layer values without pre-multiplication, and stores accumulated values in pre-multiplied form. This technique reduces the number of operations that need be performed in the loop, since the final color values are already in multiplied form.

[0052] The fade value δ_n scales the opacity of the layer as a whole. The use of the fade value is not essential to practicing the present invention; however, it facilitates effects such as fade-out or translucent dragging of layers. Fade values are known in some applications, such as PhotoShop, from Adobe Corporation.

[0053] The above description merely portrays one possible technique for merging color and alpha values in accumulator 102. Other compositing techniques may be used, as will be apparent to one skilled in the art. In one embodiment, each layer could be associated with a different compositing method, if appropriate.

[0054] It has been found that the above-described technique is able to generate an image pixel using at most N+1 read/write operations (N reads, 1 write), where N is the number of layers that can contribute to the pixel. Prior art systems generally require 3(N-1) read/write operations (2(N-1) reads, (N-1) writes). For example: if four layers A, B, C, and D are available, the present invention would perform the following read/write operations:

- Read A
- Read B
- Read C
- Read D
- Write result

[0055] Thus, where N = 4, at most N+1 = 5 operations are performed. By accumulating the result in accumulator 102, the total number of read/write operations is significantly reduced. In addition, the pixel-by-pixel top-down approach of the present invention further reduces the number of operations to be performed.

[0056] For example, one prior art technique known as the "Painter's Algorithm", and described in Foley et al., Computer Graphics: Principles and Practice, 1990, at p. 673, would composite the four layers as follows:

- Read C
- Read D
- Write result of C+D
- Read B
- Read result of C+D
- Write result of B+C+D
- Read A
- Read result of B+C+D
- Write result of A+B+C+D

[0057] Thus, where N=4, the prior art technique performs 3(N-1) = 9 read/write operations. In addition, all nine read/write operations are performed even if some of the underlying layers may not contribute to the final

result; whereas the present invention is able to reduce the number of operations when some layers may not contribute.

[0058] In addition to reducing the number of read/write operations, the present invention also reduces the number of computation operations involved, as can be seen from the above example.

[0059] In one embodiment, further optimization may be achieved as follows. System 100 can tag the topmost layer 103 that has been changed since the last time data has been written to frame buffer 104. If, in performing the front-to-back processing method described above, processor 101 stops before reaching the tagged layer 103, it is not necessary to write the new image pixel to frame buffer 104, since frame buffer 104 will already contain the same value. Thus, in a system where access to video memory is relatively slow, significant performance improvements can be attained by avoiding unnecessary writes to frame buffer 104.

[0060] In one embodiment, the invention subdivides the image area in a pre-processing step, before beginning the method described above in connection with Fig. 6. Referring now to Fig. 4, there is shown an example of image subdivision according to this embodiment.

[0061] Image 401 contains three layers 103a, 103b, and 103c to be composited. The image area can thus be subdivided into tiles 402 through 423.

For each tile, a given set of layers 103 is able to affect pixels within the tile. Thus, in the example of Fig. 4:

- tiles 402, 403, 405, 406, 410, 411, 417, 418, 421, and 423 are rendered without accessing any layers (background only);
- tiles 404, 407, and 412 are rendered using data from layer 103c;
- tiles 408 and 413 are rendered using data from layers 103b and 103c;
- tiles 409 and 419 are rendered using data from layer 103b;
- tile 414 is rendered using data from layers 103a, 103b, and 103c;
- tiles 415 and 420 are rendered using data from layers 103a and 103b; and
- tiles 416 and 422 are rendered using data from layers 103a.

[0062] Thus, each tile contains the same stack of layers to be combined in rendering the final image. In this way, most of the decision steps as to which layers may contribute to the values for a given pixel may be performed outside the innermost loop, since such decisions are constant for all pixels within a given tile. This simplifies the bounds-checking processes and improves performance, since it reduces the number of times such decision steps are performed.

[0063] Referring now to Fig. 5, there is shown an example of the subdivision technique of one embodiment of the present invention. Tile 502 represents a previously subdivided tile, or the entire image. Layer 501 is

superimposed on tile 502, resulting in subdivision of tile 502 into new tiles 503 through 507. Generalizing, this tile subdivision operation is performed as follows: for each layer 501 that has at least one side cutting through tile 502, tile 502 is subdivided by extending the top and bottom boundaries of layer 501 to the left and right edges of tile 502. Resulting tiles 503 through 507 are stored in a tile list for later retrieval by processor 101 when processing individual pixels. Some of new tiles 502 through 507 may have zero width or height, such as when layer 501 is not completely inside tile 502. If so, such degenerated rectangles are not stored in the tile list. Tile subdivision is described in more detail below, in connection with Fig. 8.

[0064] Other techniques of tile subdivision may be performed that may produce other results. It has been found that, for pixel traversal algorithms that are oriented along horizontal scan-lines, subdivision resulting in a minimal number of vertical cuts is advantageous, whereas the number of horizontal cuts is not as important.

[0065] In one embodiment, the tile list is sorted in y-order before actual compositing begins. By performing this step, the system is able to execute the update in a more orderly fashion, and reduce visible "tearing" that can occur when a number of updates are requested in quick succession.

[0066] In one embodiment, after initial tile subdivision is performed, the tile list is traversed and adjacent tiles are joined if possible, so as to reduce the

tile count for the compositing phase. Thus, in the example of Fig. 4, tiles 402, 403, 405, 406, 410, 411, 417, 418, 421, and 423 could be merged into one tile; tiles 404, 407, and 412 could be merged into a second tile; and so on. Joining tiles is described in more detail below, in connection with Fig. 9.

[0067] In one embodiment, certain tiles can be marked "frozen", indicating that they should not be further subdivided even when other layers intersect the tile. This technique serves to avoid further splitting when a tile has been created that contains a part of a completely opaque layer, since all layers that lie below the opaque one cannot be visible, so that further subdivision is not needed.

[0068] When tile subdivision is combined with the method described above in connection with Fig. 6, all pixels within a particular tile may be processed before proceeding to the next tile. In this manner, determinations as to which layers may contribute to output values may be performed at the tile level, rather than at the pixel level, thus reducing the number of operations to be performed. In addition, in one embodiment, special case algorithms may be developed for certain combinations of layers to be composited, and applied to tiles containing those combinations.

[0069] In one embodiment, as described above, several pixels may be processed simultaneously. For example, in one embodiment, eight pixels are processed in parallel, using a processor 101 that is capable of accumulating eight values simultaneously, such as AltiVec or MMX-enabled processors. Parallel

processing may also be advantageous when the invention is implemented on systems having a large number of execution units, such as the IA-64 architecture from Intel corporation.

[0070] Referring now to Fig. 8, there is shown a flowchart depicting tile subdivision according to one embodiment of the present invention. The system starts with a single tile, which represents the screen area, and then adds layers top-down by calling

```
cutTiles(struct TileInfo *tile, int x,int y, int w,int h,
    struct Layer *layer, BOOL opaque);
```

which extends the tile list as needed. The opaque flag is used to freeze tiles, as described above.

[0071] The cutTiles routine may be implemented as shown in the flowchart of Fig. 8. The steps of Fig. 8 are repeated for each existing tile in the image, so that processor 101 begins by selected 802 the next tile, if any. If the tile is frozen 803, or if the tile is completely outside 804 the area specified in the call to cutTiles, processor 101 determines 810 any more tiles exist, and proceeds 802 to the next tile. If no more tiles exist, the process ends 811.

[0072] If the tile is completely inside 805 the specified area, a new layer is created 806 corresponding to the tile, and is designated frozen if the area is opaque.

[0073] If the tile is neither completely inside or completely outside the specified area, there is overlap. Tile overlap is determined 807, and a new tile is created 808 accordingly, to represent the overlapping region. Coordinate values are assigned 809 to the new tile.

[0074] Once the tile has been processed, processor 101 determines 810 whether additional tiles exist, and if so, returns to 802 to select the next tile.

[0075] As described above, in one embodiment, after initial tile subdivision is performed, the tile list is traversed and adjacent tiles are joined if possible, so as to reduce the tile count for the compositing phase. Referring now to Fig. 9, there is shown a flowchart depicting a method of joining tiles according to one embodiment.

[0076] Processor 101 selects a tile 901 to be processed. If in 902, processor 101 determines that the tile is adjacent to another tile t having the same set of layers, the tile edges are adjusted 903 to include the area of tile t, and tile t is deleted 904. If more tiles are to be processed 905, processor 101 returns to 902.

[0077] In one embodiment, the above-described invention is used to implement a windowing system capable of displaying overlapping translucent windows in a graphical user interface. Each window is represented as one or more layers, as shown in Fig. 2. Color and alpha values associated with a window are composited as described above for overlapping tiles. When an application draws into the layer bitmap corresponding to its backing store, and requests an update on the screen, the present invention updates the screen by reassembling the corresponding portion of the frame buffer using the compositing techniques described above. Opening, closing, resizing, or moving a window triggers the tile subdivision method described above (if it is being used).

[0078] In such a windowing system, fade values as described above can be used to implement fading in and out of windows when they are opened or closed. This is accomplished by gradually changing the fade value of the layer corresponding to the window being opened or closed.

[0079] A windowing system using the above-described techniques may also implement an event routing scheme that differs from those associated with conventional overlapping windowing systems in which the topmost window at the cursor location is normally the target of a cursor-driven event. In the present system, for example, when a user performs a mouse click, the mouse-click event is sent to a particular window, which may not necessarily be the

topmost window at the cursor location. In one embodiment, the topmost non-transparent window at the cursor location is the target of the mouse event. In another embodiment, some layers may be designated as not accepting events at all; for example, a layer that contains the shadow of a window should not accept any events. As will be clear to one skilled in the art, alternative event routing schemes could also be used in connection with a windowing system as implemented according to the present invention.

[0080] From the above description, it will be apparent that the invention disclosed herein provides a novel and advantageous system and method of rendering translucent and complex-shaped layers in a display system. The foregoing discussion discloses and describes merely exemplary methods and embodiments of the present invention. As will be understood by those familiar with the art, the invention may be embodied in other specific forms without departing from the spirit or essential characteristics thereof. Accordingly, the disclosure of the present invention is intended to be illustrative, but not limiting, of the scope of the invention, which is set forth in the following claims.